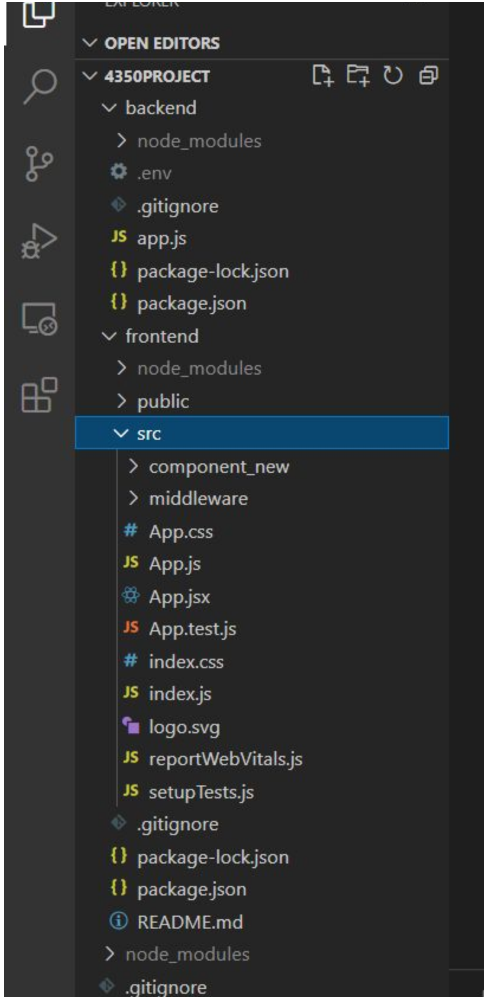
IERG 4350 Project Presentation

IP Shing On	1155109011
WONG Shing	1155109027

Project

- Voting Website
- MERN Web Application
- deploy on AWS EC2
- using nginx for React
- using process manager for Node



```
EXPLORER
∨ OPEN EDITORS
∨ 4350PROJECT
  ∨ backend
    > node_modules
      .env
      .gitignore
      app.js
      package-lock.json
      package.json
  ∨ frontend
    > node_modules
    > public
    ∨ src
      > component_new
      > middleware
        App.css
        App.js
        App.jsx
        App.test.js
        index.css
        index.js
        logo.svg
        reportWebVitals.js
        setupTests.js
      .gitignore
      package-lock.json
      package.json
      README.md
  > node_modules
    .gitignore
```

Motivation

Existing voting websites cons:

1. many registration processes before really holding a voting event
2. lack of real-time results
3. store too much private information from the users

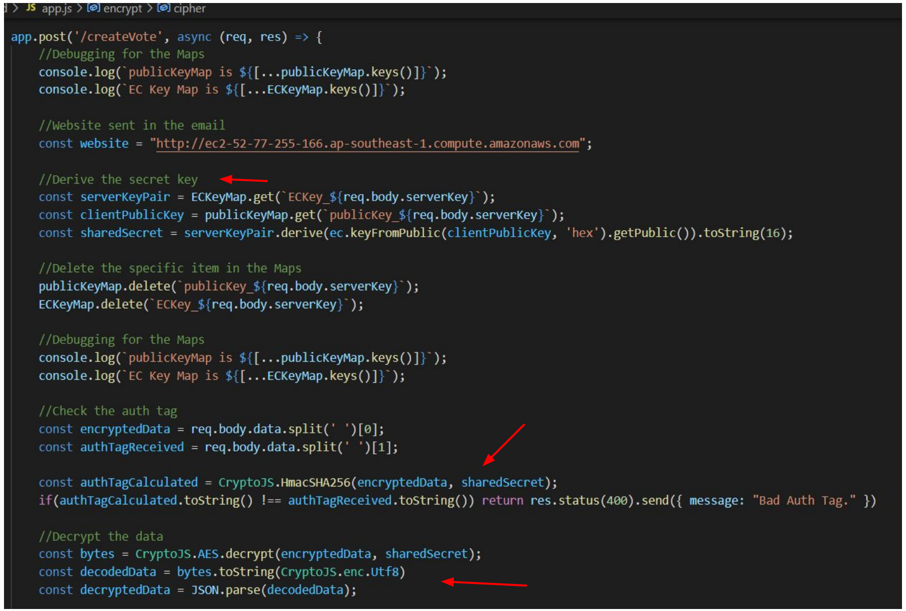
Improvement:

1. user experience
2. protect data in transit and secure database data

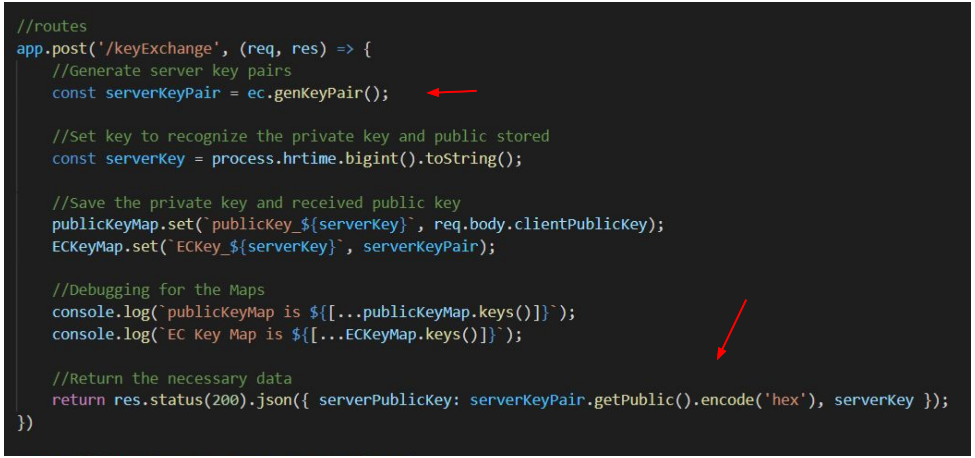
LIVE DEMO

Security Concerns

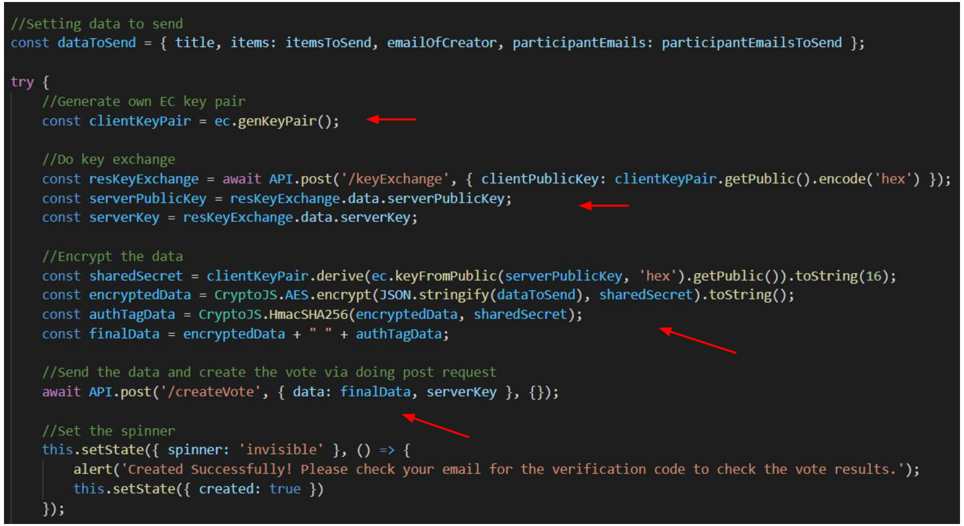
1. ECDH + AES for data in transit



```javascript
app.post('/createVote', async (req, res) => {
    //Debugging for the Maps
    console.log(`publicKeyMap is ${[...publicKeyMap.keys()]}`);
    console.log(`EC Key Map is ${[...ECKeyMap.keys()]}`);

    //Website sent in the email
    const website = "http://ec2-52-77-255-166.ap-southeast-1.compute.amazonaws.com";

    //Derive the secret key
    const serverKeyPair = ECKeyMap.get(`ECKey_${req.body.serverKey}`);
    const clientPublicKey = publicKeyMap.get(`publicKey_${req.body.serverKey}`);
    const sharedSecret = serverKeyPair.derive(ec.keyFromPublic(clientPublicKey, 'hex').getPublic()).toString(16);

    //Delete the specific item in the Maps
    publicKeyMap.delete(`publicKey_${req.body.serverKey}`);
    ECKeyMap.delete(`ECKey_${req.body.serverKey}`);

    //Debugging for the Maps
    console.log(`publicKeyMap is ${[...publicKeyMap.keys()]}`);
    console.log(`EC Key Map is ${[...ECKeyMap.keys()]}`);

    //Check the auth tag
    const encryptedData = req.body.data.split(' ')[0];
    const authTagReceived = req.body.data.split(' ')[1];

    const authTagCalculated = CryptoJS.HmacSHA256(encryptedData, sharedSecret);
    if(authTagCalculated.toString() !== authTagReceived.toString()) return res.status(400).send({ message: "Bad Auth Tag." })

    //Decrypt the data
    const bytes = CryptoJS.AES.decrypt(encryptedData, sharedSecret);
    const decodedData = bytes.toString(CryptoJS.enc.Utf8)
    const decryptedData = JSON.parse(decodedData);
```



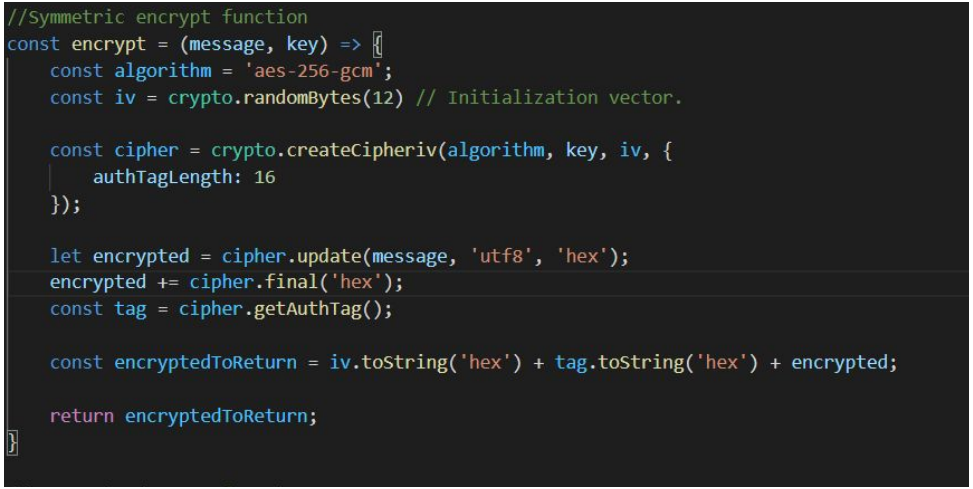
```javascript
//routes
app.post('/keyExchange', (req, res) => {
    //Generate server key pairs
    const serverKeyPair = ec.genKeyPair();

    //Set key to recognize the private key and public stored
    const serverKey = process.hrtime.bigint().toString();

    //Save the private key and received public key
    publicKeyMap.set(`publicKey_${serverKey}`, req.body.clientPublicKey);
    ECKeyMap.set(`ECKey_${serverKey}`, serverKeyPair);

    //Debugging for the Maps
    console.log(`publicKeyMap is ${[...publicKeyMap.keys()]}`);
    console.log(`EC Key Map is ${[...ECKeyMap.keys()]}`);

    //Return the necessary data
    return res.status(200).json({ serverPublicKey: serverKeyPair.getPublic().encode('hex'), serverKey });
})
```



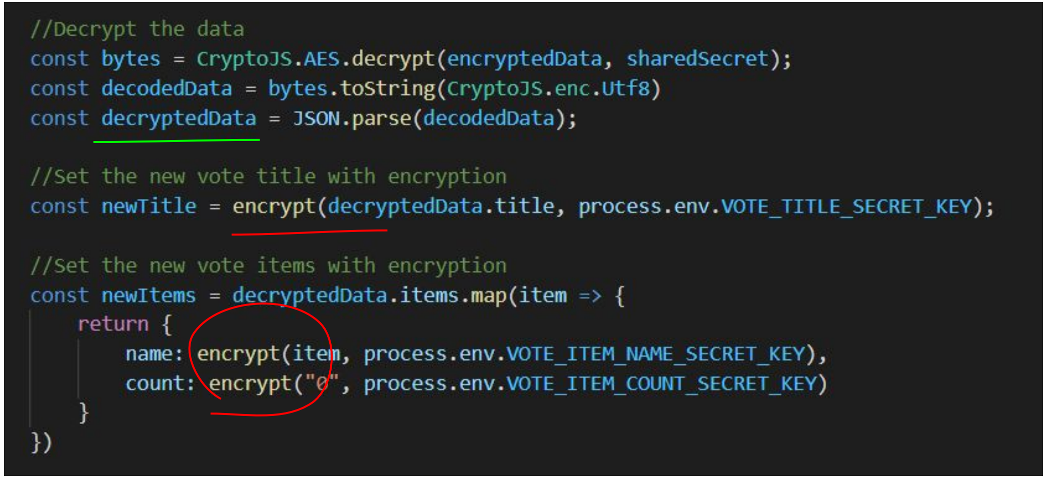
```javascript
//Setting data to send
const dataToSend = { title, items: itemsToSend, emailOfCreator, participantEmails: participantEmailsToSend };

try {
    //Generate own EC key pair
    const clientKeyPair = ec.genKeyPair();

    //Do key exchange
    const resKeyExchange = await API.post('/keyExchange', { clientPublicKey: clientKeyPair.getPublic().encode('hex') });
    const serverPublicKey = resKeyExchange.data.serverPublicKey;
    const serverKey = resKeyExchange.data.serverKey;

    //Encrypt the data
    const sharedSecret = clientKeyPair.derive(ec.keyFromPublic(serverPublicKey, 'hex').getPublic()).toString(16);
    const encryptedData = CryptoJS.AES.encrypt(JSON.stringify(dataToSend), sharedSecret).toString();
    const authTagData = CryptoJS.HmacSHA256(encryptedData, sharedSecret);
    const finalData = encryptedData + " " + authTagData;

    //Send the data and create the vote via doing post request
    await API.post('/createVote', { data: finalData, serverKey }, {});

    //Set the spinner
    this.setState({ spinner: 'invisible' }, () => {
        alert('Created Successfully! Please check your email for the verification code to check the vote results.');
        this.setState({ created: true })
    });
```

Security Concerns

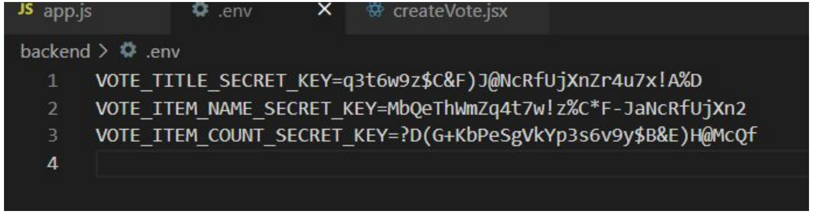
2. AES-256-GCM for data at rest



```
//Symmetric encrypt function
const encrypt = (message, key) => {
    const algorithm = 'aes-256-gcm';
    const iv = crypto.randomBytes(12) // Initialization vector.

    const cipher = crypto.createCipheriv(algorithm, key, iv, {
        authTagLength: 16
    });

    let encrypted = cipher.update(message, 'utf8', 'hex');
    encrypted += cipher.final('hex');
    const tag = cipher.getAuthTag();

    const encryptedToReturn = iv.toString('hex') + tag.toString('hex') + encrypted;

    return encryptedToReturn;
}
```



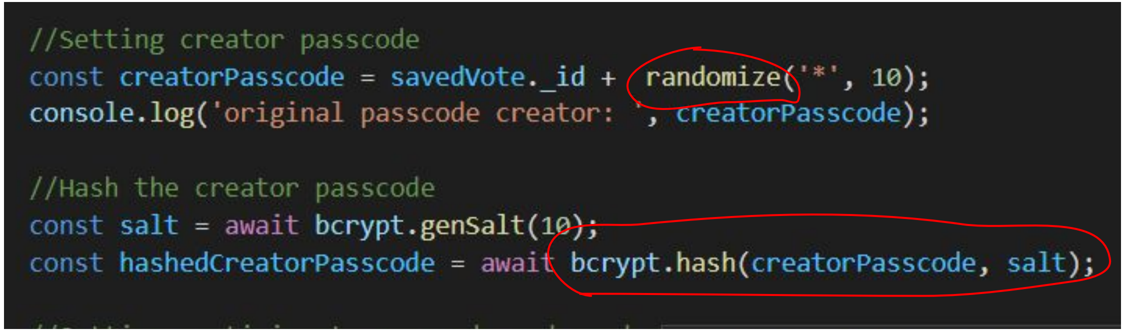
```
//Decrypt the data
const bytes = CryptoJS.AES.decrypt(encryptedData, sharedSecret);
const decodedData = bytes.toString(CryptoJS.enc.Utf8)
const decryptedData = JSON.parse(decodedData);

//Set the new vote title with encryption
const newTitle = encrypt(decryptedData.title, process.env.VOTE_TITLE_SECRET_KEY);

//Set the new vote items with encryption
const newItems = decryptedData.items.map(item => {
    return {
        name: encrypt(item, process.env.VOTE_ITEM_NAME_SECRET_KEY),
        count: encrypt("0", process.env.VOTE_ITEM_COUNT_SECRET_KEY)
    }
})
```



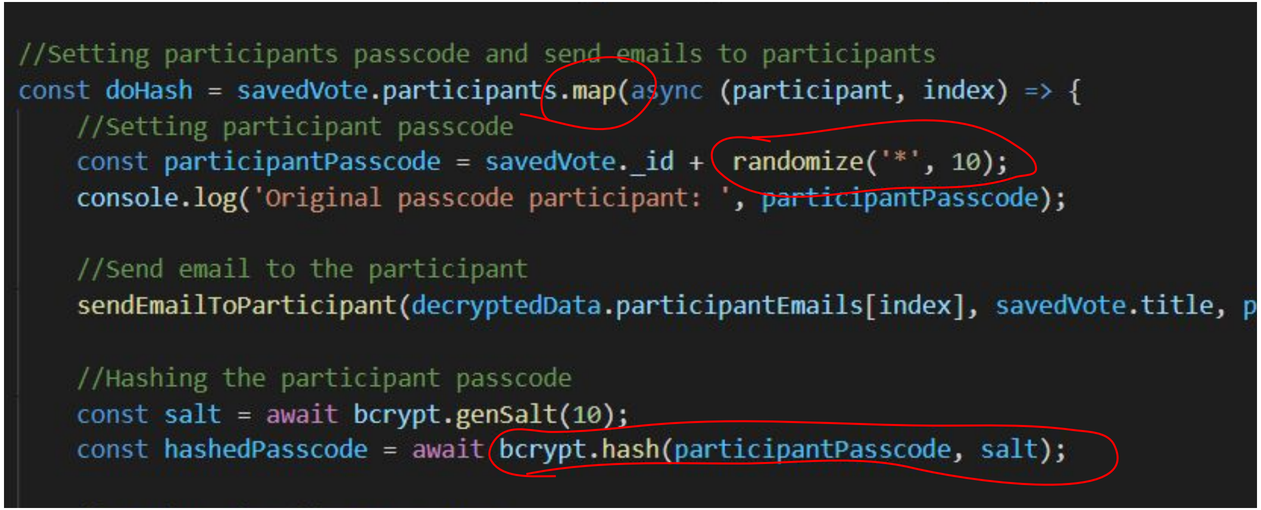
```
VOTE_TITLE_SECRET_KEY=q3t6w9z$C&F)J@NcRfUjXnZr4u7x!A%D
VOTE_ITEM_NAME_SECRET_KEY=MbQeThWmZq4t7w!z%C*F-JaNcRfUjXn2
VOTE_ITEM_COUNT_SECRET_KEY=?D(G+KbPeSgVkYp3s6v9y$B&E)H@McQf
```

Security Concerns

3. Access Validation

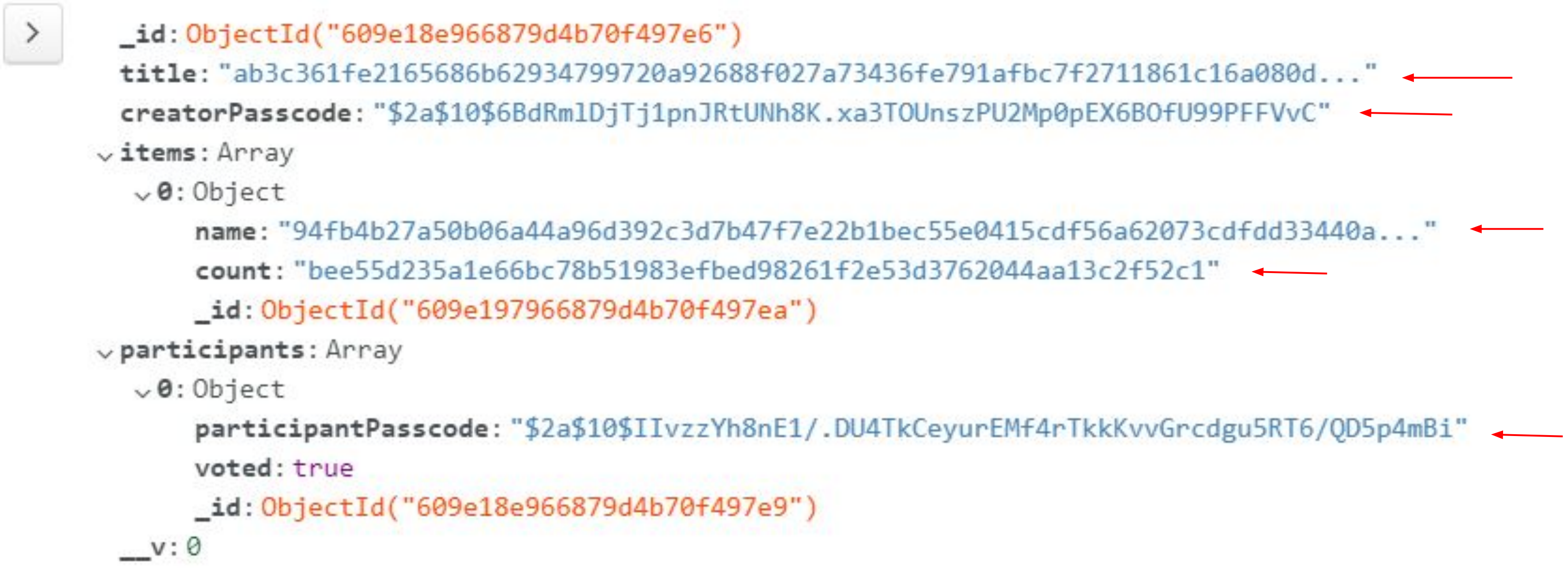


```
//Setting creator passcode
const creatorPasscode = savedVote._id + randomize('*', 10);
console.log('original passcode creator: ', creatorPasscode);

//Hash the creator passcode
const salt = await bcrypt.genSalt(10);
const hashedCreatorPasscode = await bcrypt.hash(creatorPasscode, salt);
```



```
//Setting participants passcode and send emails to participants
const doHash = savedVote.participants.map(async (participant, index) => {
    //Setting participant passcode
    const participantPasscode = savedVote._id + randomize('*', 10);
    console.log('Original passcode participant: ', participantPasscode);

    //Send email to the participant
    sendEmailToParticipant(decryptedData.participantEmails[index], savedVote.title, p

    //Hashing the participant passcode
    const salt = await bcrypt.genSalt(10);
    const hashedPasscode = await bcrypt.hash(participantPasscode, salt);
```

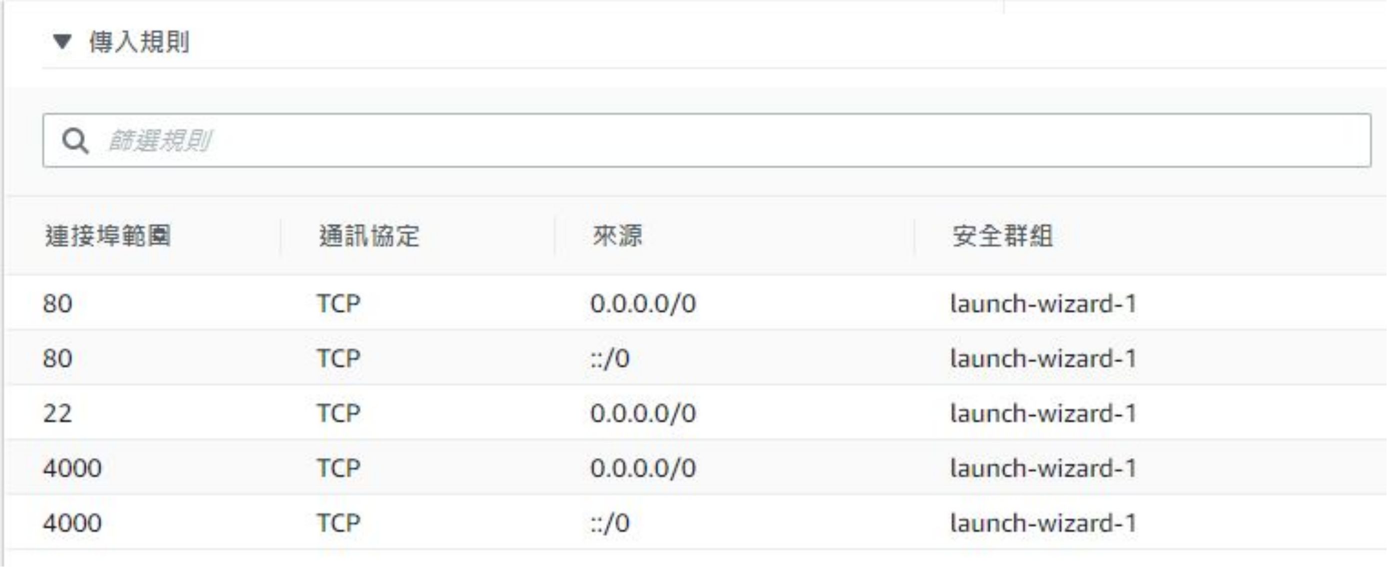
Database data



```
  _id: ObjectId("609e18e966879d4b70f497e6")
  title: "ab3c361fe2165686b62934799720a92688f027a73436fe791afbc7f2711861c16a080d..."
  creatorPasscode: "$2a$10$6BdRmlDjTj1pnJRtUNh8K.xa3TOUnszPU2Mp0pEX6BOfU99PFFVvC"
v items: Array
  v 0: Object
      name: "94fb4b27a50b06a44a96d392c3d7b47f7e22b1bec55e0415cdf56a62073cdfdd33440a..."
      count: "bee55d235a1e66bc78b51983efbed98261f2e53d3762044aa13c2f52c1"
      _id: ObjectId("609e197966879d4b70f497ea")
v participants: Array
  v 0: Object
      participantPasscode: "$2a$10$IIvzzYh8nE1/.DU4TkCeyurEMf4rTkkKvvGrcdgu5RT6/QD5p4mBi"
      voted: true
      _id: ObjectId("609e18e966879d4b70f497e9")
  __v: 0
```

Security Concerns

4. inbound rules



連接埠範圍	通訊協定	來源	安全群組
80	TCP	0.0.0.0/0	launch-wizard-1
80	TCP	::/0	launch-wizard-1
22	TCP	0.0.0.0/0	launch-wizard-1
4000	TCP	0.0.0.0/0	launch-wizard-1
4000	TCP	::/0	launch-wizard-1

Thank you!