controllers/auth.controller.js

```
const authService = require('../services/auth.service');

const authController = {
  login: (req, res, next) => {
    const { emailAddress, password } = req.body;

    authService.login(emailAddress, password, (error, data) => {
      if (error) {
        return next(error);
      }
      res.status(200).json(data);
    });
  }
};

module.exports = authController;
```

controllers/meal.controller.js

```
const mealService = require('../services/meal.service');

const mealController = {
  create: (req, res, next) => {
    const meal = req.body;
    mealService.create(meal, (error, data) => {
      if (error) {
        return next(error);
      }
      res.status(201).json(data);
    });
  },
  getAll: (req, res, next) => {
    mealService.getAll((error, data) => {
      if (error) {
        return next(error);
      }
      res.status(200).json(data);
    });
  },
  getById: (req, res, next) => {
    const mealId = req.params.mealId;
    mealService.getById(mealId, (error, data) => {
      if (error) {
        return next(error);
      }
      res.status(200).json(data);
    });
  },
  update: (req, res, next) => {
    const mealId = req.params.mealId;
    const updatedMeal = req.body;
    mealService.update(mealId, updatedMeal, (error, data) => {
      if (error) {
        return next(error);
      }
      res.status(200).json(data);
    });
  },
  delete: (req, res, next) => {
    const mealId = req.params.mealId;
    mealService.delete(mealId, (error, data) => {
      if (error) {
        return next(error);
      }
      res.status(204).json(data);
    });
  },
  participate: (req, res, next) => {
    const mealId = req.params.mealId;
    const userId = req.user.userId;
    mealService.participate(userId, mealId, (error, data) => {
      if (error) {
        return next(error);
      }
      res.status(200).json(data);
    });
  },
  cancelParticipation: (req, res, next) => {
    const mealId = req.params.mealId;
    const userId = req.user.userId;
    mealService.cancelParticipation(userId, mealId, (error, data) => {
      if (error) {
        return next(error);
      }
      res.status(200).json(data);
    });
  },
  getParticipants: (req, res, next) => {
    const mealId = req.params.mealId;
    mealService.getParticipants(mealId, (error, data) => {
      if (error) {
        return next(error);
      }
      res.status(200).json(data);
    });
  },
  getParticipantDetails: (req, res, next) => {
    const mealId = req.params.mealId;
    const participantId = req.params.participantId;
    mealService.getParticipantDetails(mealId, participantId, (error, data) =>
{
      if (error) {
        return next(error);
      }
      res.status(200).json(data);
    });
  }
};

module.exports = mealController;
```

controllers/user.controller.js

```
const userService = require('../services/user.service');

const userController = {
  create: (req, res, next) => {
    const user = req.body;
    userService.create(user, (error, data) => {
      if (error) {
        return next(error);
      }
      res.status(201).json(data);
    });
  },
  getAll: (req, res, next) => {
    userService.getAll((error, data) => {
      if (error) {
        return next(error);
      }
      res.status(200).json(data);
    });
  },
  getById: (req, res, next) => {
    const userId = req.params.userId;
    userService.getById(userId, (error, data) => {
      if (error) {
        return next(error);
      }
      res.status(200).json(data);
    });
  },
  update: (req, res, next) => {
    const userId = req.params.userId;
    const updatedUser = req.body;
    userService.update(userId, updatedUser, (error, data) => {
      if (error) {
        return next(error);
      }
      res.status(200).json(data);
    });
  },
  delete: (req, res, next) => {
    const userId = req.params.userId;
    userService.delete(userId, (error, data) => {
      if (error) {
        return next(error);
      }
      res.status(204).json(data);
    });
  },
  getProfile: (req, res, next) => {
    const userId = req.user.userId;
    userService.getProfile(userId, (error, data) => {
      if (error) {
        return next(error);
      }
      res.status(200).json(data);
    });
  }
};

module.exports = userController;
```

dao/meal.dao.js

```
const db = require('../mysql-db');

const mealDao = {
  create: (meal, callback) => {
    const query = 'INSERT INTO meal (name, description, imageUrl, price,
dateTime, maxAmountOfParticipants, isActive, isVega, isVegan, isToTakeHome,
cookId) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)';
    const values = [
      meal.name,
      meal.description,
      meal.imageUrl,
      meal.price,
      meal.dateTime,
      meal.maxAmountOfParticipants,
      meal.isActive,
      meal.isVega,
      meal.isVegan,
      meal.isToTakeHome,
      meal.cookId
    ];

    db.query(query, values, (error, result) => {
      if (error) {
        return callback(error, null);
      }
      callback(null, { id: result.insertId, ...meal });
    });
  },
  getAll: (callback) => {
    const query = 'SELECT * FROM meal';
    db.query(query, (error, result) => {
      if (error) {
        return callback(error, null);
      }
      callback(null, result);
    });
  },
  getById: (id, callback) => {
    const query = 'SELECT * FROM meal WHERE id = ?';
    db.query(query, [id], (error, result) => {
      if (error) {
        return callback(error, null);
      }
      if (result.length === 0) {
        return callback({ message: 'Meal not found' }, null);
      }
      callback(null, result[0]);
    });
  },
  update: (id, meal, callback) => {
    const query = 'UPDATE meal SET name = ?, description = ?, imageUrl = ?,
price = ?, dateTime = ?, maxAmountOfParticipants = ?, isActive = ?, isVega
= ?, isVegan = ?, isToTakeHome = ?, cookId = ? WHERE id = ?';
    const values = [
      meal.name,
      meal.description,
      meal.imageUrl,
      meal.price,
      meal.dateTime,
      meal.maxAmountOfParticipants,
      meal.isActive,
      meal.isVega,
      meal.isVegan,
      meal.isToTakeHome,
      meal.cookId,
      id
    ];

    db.query(query, values, (error, result) => {
      if (error) {
        return callback(error, null);
      }
      if (result.affectedRows === 0) {
        return callback({ message: 'Meal not found' }, null);
      }
      callback(null, { id, ...meal });
    });
  },
  delete: (id, callback) => {
    const query = 'DELETE FROM meal WHERE id = ?';
    db.query(query, [id], (error, result) => {
      if (error) {
        return callback(error, null);
      }
      if (result.affectedRows === 0) {
        return callback({ message: 'Meal not found' }, null);
      }
      callback(null, { message: 'Meal deleted successfully' });
    });
  }
};

module.exports = mealDao;
```

dao/user.dao.js

```
const db = require('../mysql-db');

const userDao = {
  create: (user, callback) => {
    const query = 'INSERT INTO user (firstName, lastName, emailAdress,
password, phoneNumber, roles, street, city) VALUES (?, ?, ?, ?, ?, ?, ?, ?)';
    const values = [
      user.firstName,
      user.lastName,
      user.emailAddress,
      user.password,
      user.phoneNumber,
      user.roles,
      user.street,
      user.city
    ];

    db.query(query, values, (error, result) => {
      if (error) {
        return callback(error, null);
      }
      callback(null, { id: result.insertId, ...user });
    });
  },
  getAll: (callback) => {
    const query = 'SELECT * FROM user';
    db.query(query, (error, result) => {
      if (error) {
        return callback(error, null);
      }
      callback(null, result);
    });
  },
  getById: (id, callback) => {
    const query = 'SELECT * FROM user WHERE id = ?';
    db.query(query, [id], (error, result) => {
      if (error) {
        return callback(error, null);
      }
      if (result.length === 0) {
        return callback({ message: 'User not found' }, null);
      }
      callback(null, result[0]);
    });
  },
  update: (id, user, callback) => {
    const query = 'UPDATE user SET firstName = ?, lastName = ?, emailAdress
= ?, password = ?, phoneNumber = ?, roles = ?, street = ?, city = ? WHERE id
= ?';
    const values = [
      user.firstName,
      user.lastName,
      user.emailAddress,
      user.password,
      user.phoneNumber,
      user.roles,
      user.street,
      user.city,
      id
    ];

    db.query(query, values, (error, result) => {
      if (error) {
        return callback(error, null);
      }
      if (result.affectedRows === 0) {
        return callback({ message: 'User not found' }, null);
      }
      callback(null, { id, ...user });
    });
  },
  delete: (id, callback) => {
    const query = 'DELETE FROM user WHERE id = ?';
    db.query(query, [id], (error, result) => {
      if (error) {
        return callback(error, null);
      }
      if (result.affectedRows === 0) {
        return callback({ message: 'User not found' }, null);
      }
      callback(null, { message: 'User deleted successfully' });
    });
  },
  getByEmail: (emailAddress, callback) => {
    const query = 'SELECT * FROM user WHERE emailAdress = ?';
    db.query(query, [emailAddress], (error, result) => {
      if (error) {
        return callback(error, null);
      }
      if (result.length === 0) {
        return callback(null, null);
      }
      callback(null, result[0]);
    });
  }
};

module.exports = userDao;
```

index.js

```
const errorMiddleware = require('./middleware/error');

const express = require('express');
const app = express();
const port = process.env.PORT || 3000;

const authRoutes = require('./routes/auth.routes');
const userRoutes = require('./routes/user.routes');
const mealRoutes = require('./routes/meal.routes');

app.use(express.json());

app.use('/api', authRoutes);
app.use('/api', userRoutes);
app.use('/api', mealRoutes);

app.get('/', (req, res) => {
  res.send('Share-a-Meal API is running!');
});

app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});

app.use(express.json());
```

middleware/auth.js

```
const jwt = require('jsonwebtoken');
const config = require('../util/config');

const authenticate = (req, res, next) => {
  const token = req.headers.authorization?.split(' ')[1];

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

  jwt.verify(token, config.secretKey, (err, decoded) => {
    if (err) {
      return res.status(403).json({ message: 'Failed to authenticate
token' });
    }

    req.user = decoded;
    next();
  });
};

module.exports = authenticate;
```

middleware/error.js

```
const logger = require('../util/logger');

const errorMiddleware = (err, req, res, next) => {
  logger.error(err.stack);

  const statusCode = err.statusCode || 500;
  const message = err.message || 'Internal Server Error';

  res.status(statusCode).json({
    status: statusCode,
    message: message,
    data: {}
  });
};

module.exports = errorMiddleware;
```

middleware/validate.js

```
const validateUser = (req, res, next) => {
    const { firstName, lastName, emailAddress, password } = req.body;

    // Validate firstName
    if (!firstName || typeof firstName !== 'string' ||
firstName.trim().length < 2) {
      return res.status(400).json({ message: 'Invalid first name' });
    }

    // Validate lastName
    if (!lastName || typeof lastName !== 'string' || lastName.trim().length <
2) {
      return res.status(400).json({ message: 'Invalid last name' });
    }

    // Validate emailAddress
    const emailRegex = /^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$/;
    if (!emailAddress || typeof emailAddress !== 'string' || !
emailRegex.test(emailAddress)) {
      return res.status(400).json({ message: 'Invalid email address' });
    }

    // Validate password
    const passwordRegex = /^(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,}$/;
    if (!password || typeof password !== 'string' || !
passwordRegex.test(password)) {
      return res.status(400).json({ message: 'Invalid password' });
    }

    next();
  };

  const validateMeal = (req, res, next) => {
    const { name, description, price, dateTime, maxAmountOfParticipants,
imageUrl } = req.body;

    // Validate name
    if (!name || typeof name !== 'string' || name.trim().length < 2) {
      return res.status(400).json({ message: 'Invalid meal name' });
    }

    // Validate description
    if (!description || typeof description !== 'string' ||
description.trim().length < 10) {
      return res.status(400).json({ message: 'Invalid meal description' });
    }

    // Validate price
    if (!price || typeof price !== 'number' || price <= 0) {
      return res.status(400).json({ message: 'Invalid meal price' });
    }

    // Validate dateTime
    if (!dateTime || !(dateTime instanceof Date) ||
isNaN(dateTime.getTime())) {
      return res.status(400).json({ message: 'Invalid meal date and time' });
    }

    // Validate maxAmountOfParticipants
    if (!maxAmountOfParticipants || typeof maxAmountOfParticipants !==
'number' || maxAmountOfParticipants < 1) {
      return res.status(400).json({ message: 'Invalid maximum number of
participants' });
    }

    // Validate imageUrl
    const imageUrlRegex = /^https?:\/\/.*\/(.+)\.(jpg|jpeg|png|gif|bmp)$/;
    if (!imageUrl || typeof imageUrl !== 'string' || !
imageUrlRegex.test(imageUrl)) {
      return res.status(400).json({ message: 'Invalid meal image URL' });
    }

    next();
  };

  module.exports = { validateUser, validateMeal };
```

mysql-db.js

```
const mysql = require('mysql2');
const logger = require('./util/logger'); // We'll create this logger utility
later
require('dotenv').config();

const dbConfig = {
  host: process.env.DB_HOST,
  port: process.env.DB_PORT,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_DATABASE,
  connectionLimit: 10,
  waitForConnections: true,
  queueLimit: 0,
  multipleStatements: true
};

const pool = mysql.createPool(dbConfig);

pool.on('connection', function (connection) {
  logger.trace(`Connected to database '${connection.config.database}' on
'${connection.config.host}:${connection.config.port}'`);
});

module.exports = pool;
```

package.json

```json
{
  "name": "shareamealfinal",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "dev": "nodemon index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "dotenv": "^16.4.5",
    "express": "^4.19.2",
    "jsonwebtoken": "^9.0.2",
    "mysql2": "^3.9.7",
    "tracer": "^1.3.0"
  },
  "devDependencies": {
    "chai": "^5.1.0",
    "chai-http": "^4.4.0",
    "mocha": "^10.4.0",
    "nodemon": "^3.1.0"
  }
}
```

routes/auth.routes.js

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

router.post('/login', authController.login);

module.exports = router;
```

routes/meal.routes.js

```
const express = require('express');
const router = express.Router();
const mealController = require('../controllers/meal.controller');

router.post('/meals', authenticate, validateMeal, mealController.create);
router.get('/meals', mealController.getAll);
router.get('/meals/:mealId', mealController.getById);
router.put('/meals/:mealId', authenticate, validateMeal,
mealController.update);
router.delete('/meals/:mealId', authenticate, mealController.delete);
router.post('/meals/:mealId/participate', authenticate,
mealController.participate);
router.delete('/meals/:mealId/participate', authenticate,
mealController.cancelParticipation);
router.get('/meals/:mealId/participants', authenticate,
mealController.getParticipants);
router.get('/meals/:mealId/participants/:participantId', authenticate,
mealController.getParticipantDetails);

module.exports = router;
```

routes/user.routes.js

```
const express = require('express');
const router = express.Router();
const userController = require('../controllers/user.controller');
const authenticate = require('../middleware/auth');

router.post('/users', validateUser, userController.create);
router.get('/users', authenticate, userController.getAll);
router.get('/users/:userId', authenticate, userController.getById);
router.put('/users/:userId', authenticate, validateUser,
userController.update);
router.delete('/users/:userId', authenticate, userController.delete);
router.get('/users/profile', authenticate, userController.getProfile);

module.exports = router;
```

services/auth.service.js

```
const jwt = require('jsonwebtoken');
const userDao = require('../dao/user.dao');
const config = require('../util/config');

const authService = {
  login: (emailAddress, password, callback) => {
    userDao.getByEmail(emailAddress, (error, user) => {
      if (error) {
        return callback(error, null);
      }

      if (!user || user.password !== password) {
        return callback({ message: 'Invalid email or password' }, null);
      }

      const token = jwt.sign({ userId: user.id }, config.secretkey,
{ expiresIn: '1h' });
      callback(null, { token });
    });
  }
};

module.exports = authService;
```

services/meal.service.js

```
const mealDao = require('../dao/meal.dao');

const mealService = {
  create: (meal, callback) => {
    mealDao.create(meal, callback);
  },
  getAll: (callback) => {
    mealDao.getAll(callback);
  },
  getById: (id, callback) => {
    mealDao.getById(id, callback);
  },
  update: (id, updatedMeal, callback) => {
    mealDao.update(id, updatedMeal, callback);
  },
  delete: (id, callback) => {
    mealDao.delete(id, callback);
  },
  participate: (userId, mealId, callback) => {
    mealDao.participate(userId, mealId, (error, data) => {
      if (error) {
        return callback(error, null);
      }
      callback(null, data);
    });
  },
  cancelParticipation: (userId, mealId, callback) => {
    mealDao.cancelParticipation(userId, mealId, (error, data) => {
      if (error) {
        return callback(error, null);
      }
      callback(null, data);
    });
  },
  getParticipants: (mealId, callback) => {
    mealDao.getParticipants(mealId, (error, data) => {
      if (error) {
        return callback(error, null);
      }
      callback(null, data);
    });
  },
  getParticipantDetails: (mealId, participantId, callback) => {
    mealDao.getParticipantDetails(mealId, participantId, (error, data) => {
      if (error) {
        return callback(error, null);
      }
      callback(null, data);
    });
  }
};

module.exports = mealService;
```

services/user.service.js

```
const userDao = require('../dao/user.dao');

const userService = {
  create: (user, callback) => {
    userDao.create(user, callback);
  },
  getAll: (callback) => {
    userDao.getAll(callback);
  },
  getById: (id, callback) => {
    userDao.getById(id, callback);
  },
  update: (id, updatedUser, callback) => {
    userDao.update(id, updatedUser, callback);
  },
  delete: (id, callback) => {
    userDao.delete(id, callback);
  },
  getProfile: (userId, callback) => {
    userDao.getById(userId, callback);
  }
};

module.exports = userService;
```

test/api.test.js

```
const chai = require('chai');
const chaiHttp = require('chai-http');
const server = require('../index');

chai.should();
chai.use(chaiHttp);

describe('Share-a-Meal API', () => {
  describe('GET /', () => {
    it('should return the welcome message', (done) => {
      chai.request(server)
        .get('/')
        .end((err, res) => {
          res.should.have.status(200);
          res.text.should.equal('Share-a-Meal API is running!');
          done();
        });
    });
  });

  // Add more test cases for API endpoints
});
```

test/meal.service.test.js

```
const chai = require('chai');
const expect = chai.expect;
const mealService = require('../services/meal.service');

describe('Meal Service', () => {
  describe('create', () => {
    it('should create a new meal', (done) => {
      const newMeal = {
        name: 'Test Meal',
        description: 'This is a test meal',
        imageUrl: 'https://example.com/test-meal.jpg',
        price: 10.99,
        dateTime: new Date(),
        maxAmountOfParticipants: 5,
        isActive: true,
        isVega: false,
        isVegan: false,
        isToTakeHome: true,
        cookId: 1
      };

      mealService.create(newMeal, (error, data) => {
        expect(error).to.be.null;
        expect(data).to.be.an('object');
        expect(data).to.have.property('id');
        expect(data.name).to.equal(newMeal.name);
        done();
      });
    });
  });

  // Add more test cases for other methods
});
```

test/user.service.test.js

```
const chai = require('chai');
const expect = chai.expect;
const userService = require('../services/user.service');

describe('User Service', () => {
  describe('create', () => {
    it('should create a new user', (done) => {
      const newUser = {
        firstName: 'John',
        lastName: 'Doe',
        emailAddress: 'john.doe@example.com',
        password: 'password123',
        phoneNumber: '1234567890',
        roles: 'guest',
        street: '123 Main St',
        city: 'Anytown'
      };

      userService.create(newUser, (error, data) => {
        expect(error).to.be.null;
        expect(data).to.be.an('object');
        expect(data).to.have.property('id');
        expect(data.firstName).to.equal(newUser.firstName);
        expect(data.lastName).to.equal(newUser.lastName);
        expect(data.emailAddress).to.equal(newUser.emailAddress);
        done();
      });
    });
  });

  describe('getAll', () => {
    it('should return an array of users', (done) => {
      userService.getAll((error, data) => {
        expect(error).to.be.null;
        expect(data).to.be.an('array');
        done();
      });
    });
  });

  // Add more test cases for other methods
});
```

util/config.js

```
const secretKey = process.env.SECRET_KEY || 'your-secret-key';

module.exports = { secretKey };
```

util/logger.js

```
const tracer = require('tracer');
require('dotenv').config();

const logger = tracer.colorConsole({
  format: '{{timestamp}} <{{title}}> {{file}}:{{line}} : {{message}}',
  dateformat: 'HH:mm:ss.L',
  level: process.env.LOG_LEVEL || 'info',
  preprocess: function (data) {
    data.title = data.title.toUpperCase();
  }
});

module.exports = logger;
```