Back-End Development Phase

Date	15 April 2025
Team ID	SWTID1744094910
Project Title	Personal Expense Tracker App
Maximum Marks	5 Marks

Back-End Logic Implementation

1. User Authentication & Authorization

Registration & Login

JWT Authentication:

- Generate JWT tokens upon successful registration/login for session management.
- Store hashed passwords using bcrypt (salt rounds: 12).

```
// Sample code for user registration
const registerUser = async (req, res) => {
  const { email, password } = req.body;
  const hashedPassword = await bcrypt.hash(password, 12);
  const user = new User({ email, password: hashedPassword });
  await user.save();
  const token = jwt.sign({ userId: user._id }, process.env.JWT_SECRET, { expiresIn: '1h' });
  res.status(201).json({ token });
};
```

Social Login (OAuth2):

- Integrate Google/Facebook login using Passport.js strategies.
- Store social IDs in the user schema for future logins.

Security Measures:

- Rate-limiting (e.g., 5 login attempts/hour) to prevent brute-force attacks.
- Account lockout after repeated failed attempts.

2. Expense Management

CRUD Operations

Create Expense:

- Validate input (amount ≥ 0, valid date, category exists).
- Auto-categorize expenses using a Naive Bayes classifier (trained on historical data).

```
// Sample expense schema
const expenseSchema = new mongoose.Schema({
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  amount: { type: Number, required: true, min: 0 },
  category: { type: String, enum: ['Food', 'Travel', 'Utilities'], required: true },
  date: { type: Date, default: Date.now },
  paymentMode: { type: String, enum: ['Cash', 'Card', 'UPI'] }
});
```

Update/Delete Expense:

- Check user ownership before allowing modifications.
- Audit logs for critical operations (e.g., deletion).

Receipt Upload

- Use multer middleware to handle file uploads.
- Store receipts in AWS S3 with unique filenames.

3. Budget Management & Alerts

Budget Setup

- Validate budget limits (e.g., monthly budget ≥ 0).
- Store budgets in a separate collection linked to users.

```
// Budget schema
const budgetSchema = new mongoose.Schema({
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  category: { type: String, required: true },
  limit: { type: Number, required: true, min: 0 },
  period: { type: String, enum: ['Daily', 'Weekly', 'Monthly'] }
});
```

Real-Time Alerts

- Use node-cron to check budgets every 6 hours:

```
cron.schedule('0 */6 * * *', async () => {
  const users = await User.find();
  users.forEach(async (user) => {
    const expenses = await Expense.find({ userId: user._id, date: { $gte: startOfMonth } });
    const total = expenses.reduce((sum, expense) => sum + expense.amount, 0);
    if (total > user.budget.limit) {
      sendAlert(user.email, 'Budget exceeded!');
    }
  });
});
```

4. Analytics & Reporting

Spending Analysis

- Use MongoDB aggregation to generate insights:

```
// Monthly spending by category
Expense.aggregate([
  { $match: { userId: req.user._id, date: { $gte: startOfMonth } } },
  { $group: { _id: "$category", total: { $sum: "$amount" } } }
]);
```

Report Generation

- Use pdf-lib to create PDF reports.
- Cache frequent report requests using Redis.

5. Security & Validation

Data Validation

- Use Joi for request body validation:

```
const expenseValidationSchema = Joi.object({
  amount: Joi.number().min(0).required(),
  category: Joi.string().valid('Food', 'Travel', 'Utilities').required()
});
```

Encryption & Compliance

- Encrypt sensitive fields (e.g., payment modes) using AES-256.
- GDPR compliance: Provide /api/user/delete endpoint for data erasure.

6. Database Design

Collections

- Users: _id, email, password, role, familyGroup.
- Expenses: _id, userId, amount, category, date, paymentMode.
- Budgets: _id, userId, category, limit, period.

Indexing

- Compound index on userId and date for faster expense queries.
- Unique index on email in the Users collection.

7. API Endpoints

Endpoint	Method	Description	Auth Required
`/api/auth/register`	POST	Register a new user	No
`/api/auth/login`	POST	Login with credentials	No
`/api/expenses`	POST	Add a new expense	Yes (JWT)
`/api/expenses/:id`	PUT	Update an expense	Yes
`/api/budgets`	POST	Set a budget	Yes
`/api/reports/monthly`	GET	Generate monthly spending report	Yes

8. Third-Party Integrations

- Twilio: Send SMS alerts for budget breaches.
- SendGrid: Email monthly reports.
- Plaid (Future): Sync bank transactions automatically.

9. Testing & Debugging

- Unit Tests: Use Mocha/Chai to test core logic (e.g., expense categorization).
- Integration Tests: Validate API endpoints with Supertest.
- Logging: Use Winston for request/error logging.

10. Deployment

- Containerization: Dockerize the backend for consistency across environments.
- CI/CD: Automate testing and deployment with GitHub Actions.
- Monitoring: Use Prometheus/Grafana for real-time performance tracking.